

Angular vs Blazor

The SPA League

David Gilson

Technical Architect

@Afelio (The NRB Group)

Patrick Grasseels

Software Engineer & Architect

@Ingenico

Qui sommes nous ?



David Gilson
Technical Architect
@ Afelio (The NRB Group)

Angular Side




Patrick Grasseels
Software Engineer
& Architect @ Ingenico

Blazor Side

Collaborators at
WeTry.tech

Angular ?



- ▶ Framework « Client Side » open-source JavaScript (TypeScript)
- ▶ Créé et maintenu par 
- ▶ Modularité = parfait pour les très grosses applications
- ▶ Éprouvé:
 - ▶ Sorti en 2016
 - ▶ Actuellement en version 8

Blazor ?



- ▶ Blazor est un framework Open source
- ▶ Il est réalisé par l'équipe ASP.NET Core chez MS
- ▶ Il est toujours en cours de développement
- ▶ Il existe en ServerSide & ClientSide
 - ▶ Server Side : toute la dynamique liée au changement des données, changement de pages, etc. se fait par SignalR
 - ▶ Client Side : toute l'application est compilée en WebAssembly
- ▶ Ça utilise la syntaxe Razor et en C#
- ▶ Il est possible de faire appel à du JS (JsInterop)

Angular vs Blazor : Quelques principes



1 - Le point d'entrée (Entry Point)

- ▶ Le point d'entrée est le premier fichier récupéré par le navigateur
- ▶ Il contient le minimum d'instruction / balise nécessaire pour lancer l'application
- ▶ Il est très similaire dans les deux Frameworks

1 - Le point d'entrée (Entry Point)



```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>AngularDemo</title>
  <base href="/" />
  <meta name="viewport" content="width=device-width">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```



```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>BlazorDemo</title>
  <base href="/" />
  <meta name="viewport" content="width=device-width" />
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app>Loading...</app>
  <script src="_framework/blazor.webassembly.js"></script>
</body>
</html>
```

2 - Les composants (Components)

- ▶ Basé sur le principe de WebComponents
 - ▶ Nouvel élément HTML
 - ▶ `<my-component/>`
- ▶ Un bouton, une partie de page allant jusqu'à l'application elle-même... sont tous des composants
- ▶ Chaque composant un **cycle de vie** facilitant l'approche événementielle (initialisation, rendu, destruction...)

2 - Les composants (Components)

Structure



- ▶ Trois (3) fichiers
 - ▶ TypeScript
 - ▶ HTML
 - ▶ CSS
- ▶ Il est possible de tout mettre en un fichier, mais ce n'est pas conseillé



- ▶ Un fichier .razor qui contient le HTML et le C#
ou
- ▶ Un fichier .razor qui contient le HTML et un fichier C# avec une classe qui contient le code C# (Code Behind)

2 - Les composants (Components)



```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-my-component',
  templateUrl: './my.component.html',
  styleUrls: ['./my.component.scss']
})
export class MyComponent implements OnInit {

  public name = 'World';

  constructor() { }

  ngOnInit() {
  }
}
```

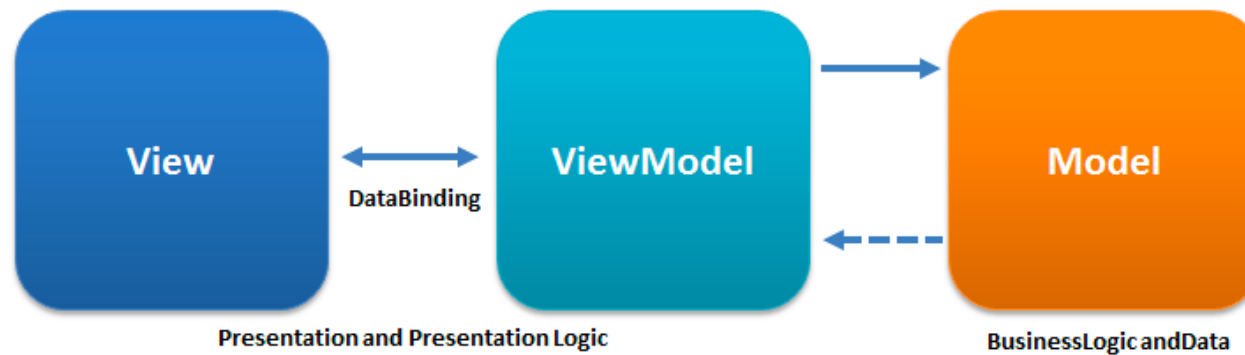
```
@page "/"
<h1>Hello @name</h1>
@code{
    private string name = "World";
}
```

```
@page "/"
@inherits PageModel
<h1>Hello @name</h1>
```

```
public class PageModel : ComponentBase {
    protected string name = "World";
}
```

3 - Le binding

- Communication entre la vue et le vue-model (MVVM)
- Communication entre les composants parent-enfant



3 - Le binding

- ▶ Afficher une valeur du C#/TS vers le HTML
- ▶ Intercepter un évènement (click, change, keyup...)
- ▶ Mettre à jour une valeur depuis le HTML vers le C#/TS

3 - Le binding

Syntaxe



- ▶ [] : property binding
- ▶ () : event binding
- ▶ [()] : binding two way
- ▶ {{}} : interpolation



- ▶ Que ce soit pour de l'event, de l'affichage ou du binding two-way tout se suffixe d'un @
 - ▶ @MaValeur
 - ▶ @bind-value="@MaValeur"
 - ▶ @onclick="MaMethode"

3 - Le binding

Incrément



```
public nbr = 0;
```

```
<button (click)="nbr = nbr + 1">
  {{ nbr }}
</button>
```



```
public int nbr = 0;
```

```
<button @onclick="@{() => nbr += 1}">
  @nbr
</button>
```

3 - Le binding



Communication parent <-> enfant



Du parent à l'enfant

```
@Input() data: string[];
```

```
<app-my-component [data]="['elem1', 'elem2']">
</app-my-component>
```

De l'enfant au parent

```
@Output() dataChange = new EventEmitter<string[]>();
```

```
this.dataChange.emit(['elem3', 'elem4']);
```

```
<app-my-component (dataChange)="dataChanged($event)">
</app-my-component>
```

Du parent à l'enfant

```
[Parameter]
public string Title { get; set; }
```

```
<ChildComponent Title="Panel Title from Parent">
```

De l'enfant au parent

```
<ChildComponent OnClick="ClickHandler"></ChildComponent>
@code{
    void ClickHandler(string newMessage)
    {
        message = newMessage;
    }
}
```

```
<button @onclick="@(() => OnClick.InvokeAsync("Hello from ChildComponent"))">
    Click me
</button>
@code{
    [Parameter] public EventCallback<string> OnClick { get; set; }
}
```

4 - Le routing

- ▶ Ajouter les bons éléments dans le DOM d'après l'URL
- ▶ Utilisation de l'API HTML5

4 - Le routing

Bases et imbrications



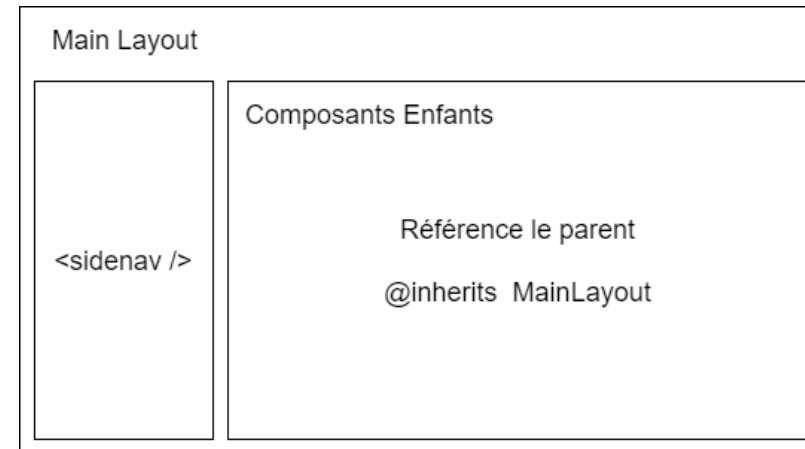
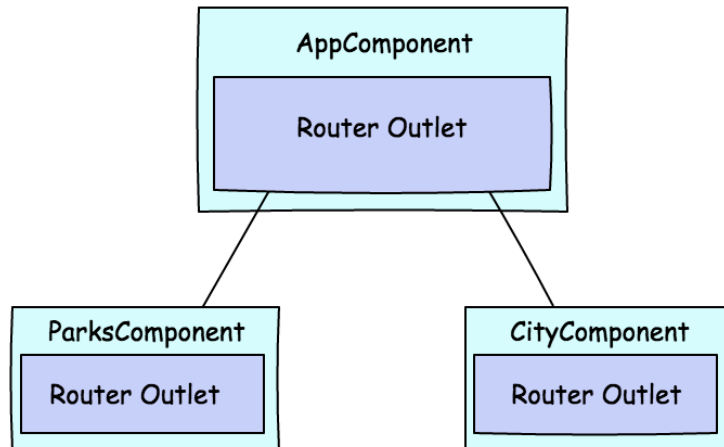
- ▶ Chaque composant est susceptible de contenir un composant enfant
- ▶ Chaque composant est susceptible d'être contenu dans un composant parent
- ▶ Ni l'enfant, ni le parent ne se connaissent l'un l'autre



- ▶ Un composant hérite ou pas d'un layout parent
- ▶ Il n'est pas obligatoire
- ▶ L'imbrication dans ce cas-ci est de l'enfant au parent, chaque enfant connaît son parent, mais le parent lui ne connaît pas ces enfants

4 - Le routing

Bases et imbrications



4 - Le routing

Bases et imbrications



```
const routes: Routes = [  
  {  
    path: 'home',  
    component: HomeComponent,  
    children: [  
      {path: 'contact', ContactComponent}  
    ]  
  }  
];
```

`<router-outlet></router-outlet>`



`@page "/maPage"`

`@Body`

4 - Le routing

Paramètres de route



```
const routes: Routes = [  
  {  
    path: 'users',  
    component: UserListComponent  
  },  
  {  
    path: 'users/:id',  
    component: UserComponent  
  }  
];
```

```
this.id = this.route.snapshot.params['id'];
```



```
@page "/MaPage/{text}"  
<h1>Hello @text</h1>
```

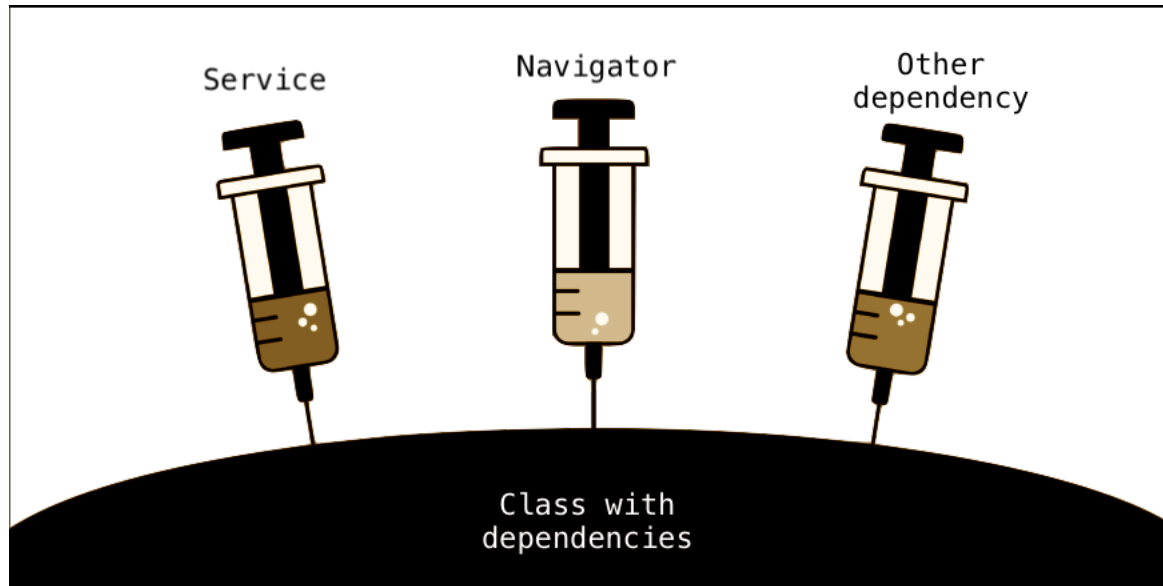
```
@code {  
  [Parameter]  
  public string Text { get; set; }  
}
```

5 - L'injection de dépendance (DI)

- ▶ Laisser le Framework gérer la création d'instances de services
- ▶ `new service1(new service2(new service3), new service4())`

5 - L'injection de dépendance (DI)

- ▶ Laisser le Framework gérer la création d'instances de services
- ▶ ~~new service1(new service2(new service3), new service4())~~



5 - L'injection de dépendance (DI)



- ▶ Logique partagée et utilisable n'importe où ailleurs dans l'application
 - ▶ Possibilité de scoper le « singleton » à l'application entière ou à un module précis
 - ▶ Utilisation via un paramètre du constructeur
- ▶ Définir des dépendances qui seront nécessaires dans nos composants
 - ▶ Services
 - ▶ Navigations
 - ▶ Validators
 - ▶ ...

5 - L'injection de dépendance (DI)



Création d'un élément injectable

```
@Injectable()  
export class LangService {  
}
```

Déclaration du service à angular

```
@NgModule({  
  providers: [LangService]  
})  
export class AppModule { }
```

Demander une instance

```
export class AppComponent implements OnInit {  
  constructor(private langService: LangService) {}  
  
  ngOnInit() {  
    this.currentLang = this.langService.currentLang;  
  }  
}
```



Création d'un élément injectable

```
@using Services  
[Inject] IDataAccess DataRepository
```

L'injection se fait comme en ASPNET Core

```
public void ConfigureServices(IServiceCollection services)  
{  
  services.AddSingleton<IDataAccess, DataAccess>();  
}
```

Demander une instance

```
[Inject]  
protected IDataAccess DataRepository { get; set; }
```


6 - Manipulation du DOM

- ▶ If = ajouter ou enlever un élément du DOM
- ▶ For = répéter une partie du DOM

6 - Manipulation du DOM



- ▶ Syntaxe particulière préfixée par une étoile « *ng... »
 - ▶ *ngIf
 - ▶ *ngFor



- ▶ Exactement la même syntaxe qu'en Razor

6 - Manipulation du DOM



```
public showP = false;
```

```
<p *ngIf="showP">I will be hidden</p>
```

```
public users = [  
  {id: 1, name: 'Patrick'},  
  {id: 2, name: 'David'},  
];
```

```
<ul>  
  <li *ngFor="let user of users">{{user.name}}</li>  
</ul>
```



```
@page "/" »  
  
@if (maCondition){  
  <h1>Hello world !</h1>  
}else{  
  <h1>Hello me !</h1>  
}  
  
@foreach(var item in items){  
  <p>@item.Name</p>  
}
```

Démystification du drag & drop



```
<ul>
  <li *ngFor="let item of items; let i=index"
      draggable="true"
      (dragstart)="dragstart(i)"
      (dragend)="dragend()"
      (dragover)="dragover(i,item)">
    {{ item }}
  </li>
</ul>
```

```
<ul>
  @foreach ((string value, Int32 i)
    in items.Select((value, i) => (value, i)))
  {
    <li draggable="true"
        @ondragstart="@(() => DragStart(i))"
        @ondragend="@(() => DragEnd())"
        @ondragover="@(() => DragOver(i, value))">
      @value (@i)
    </li>
  }
</ul>
```

Démystification du drag & drop



```
public items: string[] = [
    'Item 1',
    'Item 2',
    'Item 3'
];

private draggedElementIndex: number;

public dragstart(index: number): void {
    this.draggedElementIndex = index;
}

public dragend(): void {
    this.draggedElementIndex = null;
}

public dragover(index: number, item: string): void {
    if (index !== this.draggedElementIndex) {
        const tmp: string =
            this.items[this.draggedElementIndex];
        this.items[this.draggedElementIndex] = item;
        this.items[index] = tmp;
        this.draggedElementIndex = index;
    }
}
```



```
@code{
    private List<string> items = new List<string>()
    {
        "Item 1",
        "Item 2",
        "Item 3"
    };

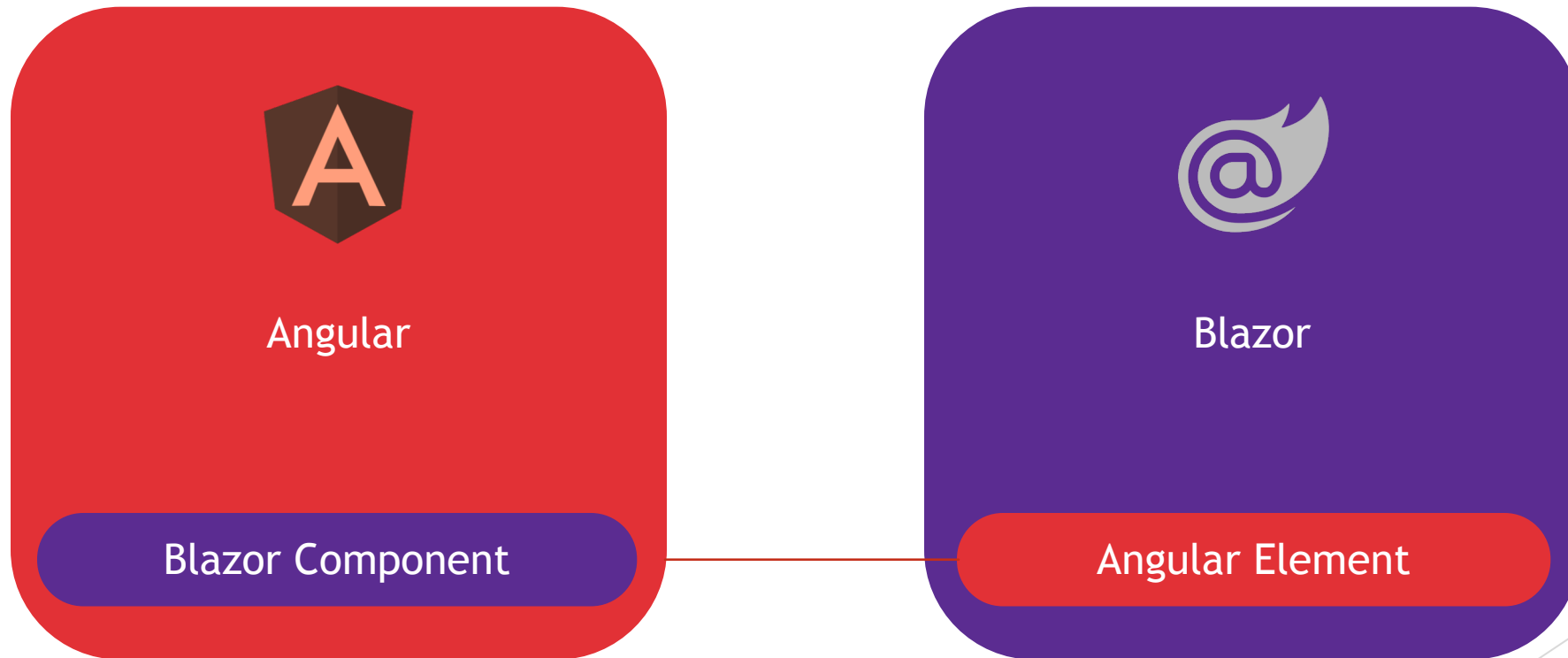
    private int? draggedElementIndex;

    public void DragStart(int index)
    {
        this.draggedElementIndex = index;
    }

    public void DragEnd()
    {
        this.draggedElementIndex = null;
    }

    public void DragOver(int index, string item)
    {
        if (index != draggedElementIndex)
        {
            var tmp = items[(int)draggedElementIndex];
            items[(int)draggedElementIndex] = item;
            items[index] = tmp;
            draggedElementIndex = index;
        }
    }
}
```

Et si ... ?



Conclusion

- ▶ On n'est pas « concurrent »
- ▶ On fait la même chose, mais différemment tout en gardant des concepts de base similaires
 - ▶ Blazor en C#/Razor
 - ▶ Angular en Typescript
- ▶ Blazor ouvre aux personnes qui n'ont jamais fait de Frontend, de réaliser avec leurs connaissances en C# des SPA.

Merci Questions ?

